

Request Complexity of VNet Topology Extraction: Dictionary-Based Attacks^{*}

Yvonne-Anne Pignolet¹, Stefan Schmid², Gilles Tredan³

¹ ABB Corporate Research, Switzerland

² Telekom Innovation Laboratories & TU Berlin, Germany

³ CNRS-LAAS, France

Abstract. The network virtualization paradigm envisions an Internet where arbitrary virtual networks (VNet) can be specified and embedded over a shared substrate (e.g., the physical infrastructure). As VNet can be requested at short notice and for a desired time period only, the paradigm enables a flexible service deployment and an efficient resource utilization.

This paper investigates the security implications of such an architecture. We consider a simple model where an attacker seeks to extract secret information about the substrate topology, by issuing repeated VNet embedding requests. We present a general framework that exploits basic properties of the VNet embedding relation to infer the entire topology. Our framework is based on a graph motif dictionary applicable for various graph classes. We provide upper bounds on the *request complexity*, the number of requests needed by the attacker to succeed. Moreover, we present some experiments on existing networks to evaluate this dictionary-based approach.

1 Introduction

While network virtualization enables a flexible resource sharing, opening the infrastructure for automated virtual network (VNet) embeddings or service deployments may introduce new kinds of security threats. For example, by virtualizing its network infrastructure (e.g., the links in the aggregation or backbone network, or the computational or storage resources at the points-of-presence), an Internet Service Provider (ISP) may lose control over how its network is used. Even if the ISP manages the allocation and migration of VNet slices and services itself and only provides a very rudimentary interface to interact with customers (e.g., service or content providers), an attacker may infer information about the network topology (and state) by generating VNet requests.

This paper builds upon the model introduced in [11] and studies complexity of the *topology extraction problem*: How many VNet requests are required to infer the full topology of the infrastructure network? While algorithms for trees and cactus graphs with request complexity $O(n)$ and a lower bound for general graphs of $\Omega(n^2)$ have been shown in [11], graph classes between these extremes have not been studied.

Contribution. This paper presents a general framework to solve the topology extraction problem. We first describe necessary and sufficient conditions which facilitate

^{*} This project was partly funded by the Secured Virtual Cloud (SVC) project.

the “greedy” exploration of the substrate topology (the *host graph* H) by iteratively extending the requested VNet graph (the *guest graph* G). Our framework then exploits these conditions to construct an ordered (request) *dictionary* defined over so-called *graph motifs*. We show how to apply the framework to different graph families, discuss the implications on the request complexity, and also report on a small simulation study on realistic topologies. These empirical results show that many scenarios can indeed be captured with a small dictionary, and small motifs are sufficient to infer if not the entire, then at least a significant fraction of the topology.

2 Background

This section presents our model and discusses how it compares to related work.

Model. The VNet embedding based topology extraction problem has been introduced in [11]. The formal setting consists of two entities: a *customer* (the “adversary”) that issues virtual network (VNet) requests and a *provider* that performs the access control and the embedding of VNets. We model the virtual network requests as simple, undirected graphs $G = (V, E)$ (the *guest graph*) where V denotes the virtual nodes and E denotes the virtual edges connecting nodes in V . Similarly, the infrastructure network is given as an undirected graph $H = (V, E)$ (the so-called *host graph* or *substrate*) as well, where V denotes the set of substrate nodes, E is the set of substrate links, and w is a capacity function describing the available resources on a given node or edge. Without loss of generality, we assume that H is connected and that there are no parallel edges or self-loops neither in VNet requests nor in the substrate.

In this paper we assume that besides the resource demands, the VNet requests do not impose any mapping restrictions, i.e., a virtual node can be mapped to *any* substrate node, and we assume that a virtual link connecting two substrate nodes can be mapped to an entire (but single) *path* on the substrate as long as the demanded capacity is available. These assumptions are typical for virtual networks [5].

A virtual link which is mapped to more than one substrate link however can entail certain costs at the *relay nodes*, the substrate nodes which do not constitute endpoints of the virtual link and merely serve for forwarding. We model these costs with a parameter $\epsilon > 0$ (per link). Moreover, we also allow multiple virtual nodes to be mapped to the same substrate node if the node capacity allows it; we assume that if two virtual nodes are mapped to the same substrate node, the cost of a virtual link between them is zero.

Definition 1 (Embedding π , Relation \mapsto). An embedding of a graph $A = (V_A, E_A, w_A)$ to a graph $B = (V_B, E_B, w_B)$ is a mapping $\pi : A \rightarrow B$ where every node of A is mapped to exactly one node of B , and every edge of A is mapped to a path of B . That is, π consists of a node $\pi_V : V_A \rightarrow V_B$ and an edge mapping $\pi_E : E_A \rightarrow P_B$, where P_B denotes the set of paths. We will refer to the set of virtual nodes embedded on a node $v_B \in V_B$ by $\pi_V^{-1}(v_B)$; similarly, $\pi_E^{-1}(e_B)$ describes the set of virtual links passing through $e_B \in E_B$ and $\pi_E^{-1}(v_B)$ describes the virtual links passing through $v_B \in V_B$ with v_B serving as a relay node.

To be valid, the embedding π has to fulfill the following properties: (i) Each node $v_A \in V_A$ is mapped to exactly one node $v_B \in V_B$ (but given sufficient capacities, v_B can host multiple nodes from V_A). (ii) Links are mapped consistently, i.e., for two

nodes $v_A, v'_A \in V_A$, if $e_A = \{v_A, v'_A\} \in E_A$ then e_A is mapped to a single (possibly empty and undirected) path in B connecting nodes $\pi(v_A)$ and $\pi(v'_A)$. A link e_A cannot be split into multiple paths. (iii) The capacities of substrate nodes are not exceeded: $\forall v_B \in V_B: \sum_{u \in \pi_V^{-1}(v_B)} w(u) + \epsilon \cdot |\pi_E^{-1}(v_B)| \leq w(v_B)$. (iv) The capacities in E_B are respected as well, i.e., $\forall e_B \in E_B: \sum_{e \in \pi_E^{-1}(e_B)} w(e) \leq w(e_B)$.

If there exists such a valid embedding mapping π , we say that graph A can be embedded in B , denoted by $A \mapsto B$. Hence, \mapsto denotes the VNet embedding relation.

The provider has a flexible choice where to embed a VNet as long as a valid mapping is chosen. In order to design topology discovery algorithms, we exploit the following property of the embedding relation.

Lemma 1. *The embedding relation \mapsto applied to any family \mathcal{G} of undirected graphs (short: (\mathcal{G}, \mapsto)), forms a partially ordered set (a poset). [Proof in Appendix]*

We are interested in algorithms that “guess” the target topology H (the host graph) among the set \mathcal{H} of possible substrate topologies. Concretely, we assume that given a VNet request G (a guest graph), the substrate provider always responds with an *honest (binary) reply* R informing the customer whether the requested VNet G is embeddable on the substrate H . Based on this reply, the attacker may then decide to ask the provider to embed the corresponding VNet G on H , or it may not embed it and continue asking for other VNets. Let ALG be an algorithm asking a series of requests G_1, \dots, G_t to reveal H . The *request complexity* to infer the topology is measured in the number of requests t (in the worst case) until ALG issues a request G_t which is isomorphic to H and *terminates* (i.e., ALG knows that $H = G_t$ and does not issue further requests).

Related Work. Embedding VNets is an intensively studied problem and there exists a large body of literature (e.g., [7,9,12,14]), also on distributed computing approaches [8] and online algorithms [3,6]. Our work is orthogonal to this line of literature in the sense that we assume that an (arbitrary and not necessarily resource-optimal) embedding algorithm is *given*. Instead, we focus on the question of how the feedback obtained through these algorithms can be exploited, and we study the implications on the information which can be obtained about a provider’s infrastructure.

Our work studies a new kind of topology inference problem. Traditionally, much graph discovery research has been conducted in the context of today’s complex networks such as the Internet which have fascinated scientists for many years, and there exists a wealth of results on the topic. The classic instrument to discover Internet topologies is *traceroute* [4], but the tool has several problems which makes the problem challenging. One complication of traceroute stems from the fact that routers may appear as stars (i.e., anonymous nodes), which renders the accurate characterization of Internet topologies difficult [1,10,13]. *Network tomography* is another important field of topology discovery. In network tomography, topologies are explored using pairwise end-to-end measurements, without the cooperation of nodes along these paths. This approach is quite flexible and applicable in various contexts, e.g., in social networks. For a good discussion of this approach as well as results for a routing model along shortest and second shortest paths see [2]. For example, [2] shows that for sparse random graphs, a relatively small number of cooperating participants is sufficient to discover a network

fairly well. Both the traceroute and the network tomography problems differ from our virtual network topology discovery problem in that the exploration there is inherently *path-based* while we can ask for entire virtual graphs.

The paper closest to ours is [11]. It introduces the topology extraction model studied in this paper, and presents an asymptotically optimal algorithm for the cactus graph family (request complexity $\Theta(n)$), as well as a general algorithm (based on spanning trees) with request complexity $\Theta(n^2)$.

3 Motif-Based Dictionary Framework

The algorithms for tree and cactus graphs presented in [11] can be extended to a framework for the discovery of more general graph classes. It is based on the idea of growing sequences of subgraphs from nodes discovered so far. Intuitively, in order to describe the “knitting” of a given part of a graph, it is often sufficient to use a small set of graph *motifs*, without specifying all the details of how many substrate nodes are required to realize the motif. We start this section with the introduction of motifs and their composition and expansion. Then we present the dictionary concept, which structures motif sequences in a way that enables the efficient host graph discovery with algorithm DICT. Subsequently, we give some examples and finally provide the formal analysis of the request complexity.

3.1 Motifs: Composition and Expansion

In order to define the motif set of a graph family \mathcal{H} , we need the concept of *chain (graph) C*: C is just a graph $G = (\{v_1, v_2\}, \{v_1, v_2\})$ consisting of two nodes and a single link. As its edge represents a virtual link that may be embedded along entire path in the substrate network, it is called a *chain*.

Definition 2 (Motif). *Given a graph family \mathcal{H} , the set of motifs of \mathcal{H} is defined constructively: If any member of $H \in \mathcal{H}$ has an edge cut of size one, the chain C is a motif for \mathcal{H} . All remaining motifs are at least 2-connected (i.e., any pair of nodes in a motif is connected by at least two vertex-disjoint paths). These motifs can be derived by the at least 2-connected components of any $H \in \mathcal{H}$ by repeatedly removing all nodes with degree smaller or equal than two from H (such nodes do not contribute to the knitting) and merging the incident edges, as long as all remaining cycles do not contain parallel edges. Only one instance of isomorphic motifs is kept.*

Note that the set of motifs of \mathcal{H} can also be computed by iteratively by removing all low-degree nodes and subsequently determine the graphs connecting nodes constituting a vertex-cut of size one for each member $H \in \mathcal{H}$. In other words, the motif set \mathcal{M} of a graph family \mathcal{H} is a set of non-isomorphic minimal (in terms of number of nodes) graphs that are required to construct each member $H \in \mathcal{H}$ by taking a motif and either replacing edges with two edges connected by a node or gluing together components several times. More formally, a graph family containing all elements of \mathcal{H} can be constructed by applying the following rules repeatedly.

Definition 3 (Rules). (1) Create a new graph consisting of a motif $M \in \mathcal{M}$ (New Motif Rule). (2) Given a graph created by these rules, replace an edge e of H by a new node and two new edges connecting the incident nodes of e to the new node (Insert Node Rule). (3) Given two graphs created by these rules, attach them to each other such that they share exactly one node (Merge Rule).

Being the inverse operations of the ones to determine the motif set, these rules are sufficient to compose all graphs in \mathcal{H} : If \mathcal{M} includes all motifs of \mathcal{H} , it also includes all 2-connected components of H , according to Definition 2. These motifs can be glued together using the *Merge Rule*, and eventually the low-degree nodes can be added using the *Insert Node Rule*. Therefore, we have the following lemma.

Lemma 2. *Given the motifs \mathcal{M} of a graph family \mathcal{H} , the repeated application of the rules in Definition 3 allows us to construct each member $H \in \mathcal{H}$.*

However, note that it may also be possible to use these rules to construct graphs that are *not* part of the family. The following lemma shows that when degree-two nodes are added to a motif M to form a graph G , all network elements (substrate nodes and links) are *used* when embedding M in G (i.e., $M \mapsto G$).

Lemma 3. *Let $M \in (\mathcal{M} \setminus \{C\})$ be an arbitrary two-connected motif, and let G be a graph obtained by applying the Insert Node Rule (Rule 2 of Definition 3) to motif M . Then, an embedding $M \mapsto G$ involves all nodes and edges in G : at least ϵ resources are used on all nodes and edges.*

Proof. Let $v \in G$. Clearly, if there exists $u \in M$ such that $v = \pi(u)$, then v 's capacity is used fully. Otherwise, v was added by Rule 2. Let a, b be the two nodes of G between which Rule 2 was applied, and hence $\{\pi^{-1}(a), \pi^{-1}(b)\} \in E_M$ must be a motif edge. Observe that for these nodes' degrees it holds that $\deg(a) = \deg(\pi^{-1}(a))$ and $\deg(b) = \deg(\pi^{-1}(b))$ since Rule 2 never modifies the degree of the old nodes in the host graph G . Since links are of unit capacity, each substrate link can only be used once: at a at most $\deg(a)$ edge-disjoint paths can originate, which yields a contradiction to the degree bound, and the relaying node v has a load of ϵ . \square

Lemma 3 implies that no additional nodes can be inserted to an existing embedding. In other words, a motif constitutes a “minimal reservation pattern”. As we will see, our algorithm will exploit this invariant that motifs cover the entire graph knitting, and adds simple nodes (of degree 2) only in a later phase.

Corollary 1. *Let $M \in (\mathcal{M} \setminus \{C\})$ and let G be a graph obtained by applying Rule 2 of Definition 3 to motif M . Then, no additional node can be embedded on G after embedding $M \mapsto G$.*

Next, we want to *combine* motifs explore larger “knittings” of graphs. Each motif pair is glued together at a single node *or* edge (“attachment point”): We need to be able to conceptually join to motifs at edges as well because the corresponding edge of the motif can be expanded by the *Insert Node Rule* to create a node where the motifs can be joined.

Definition 4 (Motif Sequences, Subsequences, Attachment Points, \prec). A motif sequence S is a list $S = (M_1 a_1 a'_1 M_2 \dots M_k)$ where $\forall i : M_i \in \mathcal{M}$ and where M_i is glued together at exactly one node with M_{i-1} (i.e., M_i is “attached” to a node of motif M_{i-1}): the notation $M_{i-1} a_{i-1} a'_{i-1} M_i$ specifies the selected attachment points a_{i-1} and a'_{i-1} . If the attachment points are irrelevant, we use the notation $S = (M_1 M_2 \dots M_k)$ and M_i^k denotes an arbitrary sequence consisting of k instances of M_i . If S can be decomposed into $S = S_1 S_2 S_3$, where S_1, S_2 and S_3 are (possibly empty) motif sequences as well, then S_1, S_2 and S_3 are called subsequences of S , denoted by \prec .

In the following, we will sometimes use the *Kleene star* notation X^* to denote a sequence of (zero or more) elements of X attached to each other.

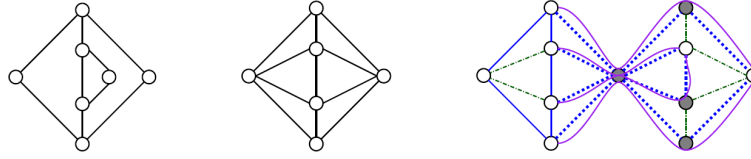


Fig. 1. *Left:* Motif A. *Center:* Motif B. Observe that $A \not\rightarrow B$. *Right:* Motif A is embedded into two consecutive Motifs B: solid lines are virtual links mapped on single substrate links, solid curves are virtual links mapped on multiple substrate links, dotted lines are substrate links implementing a multi-hop virtual link, and dashed lines are substrate unused links. Grayed nodes are relay-only nodes. Observe that the central node has a relaying load of 4ϵ .

One has to be careful when arguing about the embedding of motif sequences, as illustrated in Figure 1 which shows a counter example for $M_i \not\rightarrow M_j \Rightarrow \forall k > 0, M_i \not\rightarrow M_j^k$. This means that we typically cannot just incrementally add motif occurrences to discover a certain substructure. This is the motivation for introducing the concept of a *dictionary* which imposes an order on motif sequences and their attachment points.

3.2 Dictionary Structure and Existence

In a nutshell, a dictionary is a *Directed Acyclic Graph (DAG)* defined over all possible motifs \mathcal{M} . and imposes an order (poset relationship \mapsto) on problematic motif sequences which need to be embedded one before the other (e.g., the composition depicted in Figure 1). To distinguish them from sequences, dictionary entries are called *words*.

Definition 5 (Dictionary, Words). A dictionary $D(V_D, E_D)$ is a directed acyclic graph (DAG) over a set of motif sequences V_D together with their attachment points. In the context of the dictionary, we will call a motif sequence word. The links E_D represent the poset embedding relationship \mapsto .

Concretely, the DAG has a single root r , namely the chain graph C (with two attachment points). In general, the attachment points of each vertex $v \in V_D$ describing a word w define how w can be connected to other words. The directed edges

$E_D = (v_1, v_2)$ represent the transitively reduced embedding poset relation with the chain C context: Cv_1C is embeddable in Cv_2C and there is no other word Cv_3C such that $Cv_1C \mapsto Cv_3C$, $Cv_3C \mapsto Cv_2C$ and $Cv_3C \not\mapsto Cv_1C$ holds. (The chains before and after the words are added to ensure that attachment points are “used”: there is no edge between two isomorphic words with different attachment point pairs.)

We require that the dictionary be robust to composition: For any node v , let $R_v = \{v' \in V_D, v \mapsto v'\}$ denote the “reachable” set of words in the graph and $\bar{R}_v = V_D \setminus R_v$ all other words. We require that $v \not\mapsto W, \forall W \in Q_i := \bar{R}_i^* \setminus R_i^*$, where the transitive closure operator X^* denotes an arbitrary sequence (including the empty sequence) of elements in X (according to their attachment points).

See Figure 2 for an example. Informally, the robustness requirement means that the word represented by v cannot be embedded in any sequence of “smaller” words, unless a subsequence of this sequence is in the dictionary as well. As an example, in a dictionary containing motifs A and B from Figure 1 would contain vertices A , B and also BB , and a path from A to BB . In the following, we use the notation $\max_{v \in V_D} (v \mapsto S)$

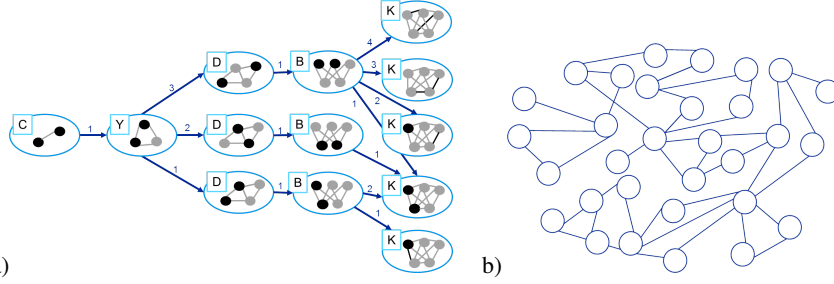


Fig. 2. a) Example dictionary with motifs Chain C , Cycle Y , Diamond D , complete bipartite graph $B = K_{2,3}$ and complete graph $K = K_5$. The attachment point pair of each word is black, the other nodes and edges of the words are grey. The edges of the dictionary are locally labeled, which is used in DICT later. b) A graph that can be constructed from the dictionary words.

to denote the set of “maximal” vertices with respect to their embeddability into S : $i \in \max_{v \in V_D} (v \mapsto S) \Leftrightarrow (i \mapsto S) \wedge (\forall j \in \Gamma^+(i), j \not\mapsto S)$, where $\Gamma^+(v)$ denotes the set of outgoing neighbors of v . Furthermore, we say that a dictionary D covers a motif sequence S iff S can be formed by concatenating dictionary words (henceforth denoted by $S \in D^*$) at the specified attachment points. More generally, a dictionary covers a graph, if it can be formed by merging sequences of D^* .

Let us now derive some properties of the dictionary which are crucial for a proper substrate topology discovery. First we consider maximal dictionary words which can serve as embedding “anchors” in our algorithm.

Lemma 4. Let D be a dictionary covering a sequence S of motifs, and let $i \in \max_{v \in V_D} (v \mapsto S)$. Then i constitutes a subsequence of S , i.e., S can be decomposed to $S_1 i S_2$, and S contains no words of order at most i , i.e., $S_1, S_2 \in (\bar{R}_i \cup \{i\})^*$.

Proof. By contradiction assume $i \in \max_{v \in V_D} (v \mapsto S)$ and i is not a subsequence of S (written $i \not\mapsto S$). Since D covers S we have $S \in V_D^*$ by definition.

Since D is a dictionary and $i \mapsto S$ we know that $S \notin Q_i$. Thus, $S \in D^* \setminus Q_i$: S has a subsequence of at least one word in R_i . Thus there exists $k \in R_i$ such that $k \prec S$. If $k = i$ this implies $i \prec S$ which contradicts our assumption. Otherwise it means that $\exists j \in \Gamma^+(i)$ such that $j \mapsto k \prec S$, which contradicts the definition of $i \in \max_{v \in V_D}(v \mapsto S)$ and thus it must hold that $i \prec S$. \square

The following corollary is a direct consequence of the definition of $i \in \max_{v \in V_D}(v \mapsto S)$ and Lemma 4: since for a motif sequence S with $S \in (\overline{R}_i \cup \{i\})^*$, all the subsequences of S that contain no i are in \overline{R}_i^* . As we will see, the corollary is useful to identify the motif words composing a graph sequence, from the most complex words to the least complex ones.

Corollary 2. *Let D be a dictionary covering a motif sequence S , and let $i \in \max_{v \in V_D}(v \mapsto S)$. Then S can be decomposed as a sequence $S = T_1 i T_2 i, \dots, i T_k$ with $T_j \in Q_i, \forall j = 1, \dots, k$.*

This corollary can be applied recursively to describe a motif sequence as a sequence of dictionary entries. Note that a dictionary always exists.

Lemma 5. *There exists a dictionary $D = (V_D, E_D)$ that covers all member graphs H of a motif graph family \mathcal{H} with n vertices. [Proof in Appendix]*

3.3 The Dictionary Algorithm

With these concepts in mind, we are ready to describe our generalized graph discovery algorithm called DICT (cf Algorithm 1). Basically, DICT always grows a request graph $G = H'$ until it is isomorphic to H (the graph to be discovered). This graph growing is performed according to the dictionary, i.e., we try to embed new motifs in the order imposed by the dictionary DAG.

DICT is based on the observation that it is very costly to discover additional edges between nodes in a 2-connected component: essentially, finding a single such edge requires testing all possibilities, which is quadratic in the component size. Thus, it is crucial to first explore the basic “knitting” of the topology, i.e., the minors which are at least 2-connected (the *motifs*). In other words, we maintain the invariant that there are never two nodes u, v which are not k -connected in the currently requested graph H' while they are k -connected in H ; no path relevant for the connectivity is overlooked and needs to be found later.

Nodes and edges which are not contributing to the connectivity need not be explored at this stage yet, as they can be efficiently added later. Concretely, these additional nodes can then be discovered by (1) using an *edge expansion* (where additional degree two nodes are added along a motif edge), and by (2) adding “chains” C to the nodes (a virtual link C constitutes an edge cut of size one and can again be expanded to entire chain of nodes using *edge expansion*).

Let us specify the *topological order* in which algorithm DICT discovers the dictionary words. First, for each node v in V_D , we define an order on its outgoing edges $\{(v, w) | w \in \Gamma^+(v)\}$. This order is sometimes referred to as a “port labeling”, and each path from the dictionary root (the chain C) to a node in V_D can be represented

as the sequence of port labels at each traversed node (l_1, l_2, \dots, l_l) , where l_1 corresponds to a port number in C . We can simply use the lexicographic order on integers, $<^d: (a_1, a_2, \dots, a_{n_1}) <^d (b_1, b_2, \dots, b_{n_2}) \iff ((\exists m > 0) (\forall i < m)(a_i = b_i) \wedge (a_m < b_m)) \vee (\forall i \in \{1, \dots, n_1\}, (a_i = b_i) \wedge (n_1 < n_2))$, to associate each vertex with its minimal sequence, and sort vertices of V_D according to their embedding order. Let r be the *rank* function associating each vertex with its position in this sorting: $r : V_D \rightarrow \{1, \dots, |V_D|\}$ (i.e., r is the topological ordering of D).

The fact that subsequences can be defined recursively using a dictionary (Lemma 4 and Corollary 2) is exploited by algorithm DICT. Concretely, we apply Corollary 2 to gradually identify the words composing a graph sequence, from the most complex words to the least complex ones. This is achieved by traversing the dictionary depth-first, starting from the root C up to a maximal node: algorithm DICT tests the nodes of $I^+(v)$ in increasing port order as defined above. As a shorthand, the word $v \in V_D$ with $r(v) = i$ is written as $D[i]$; similarly $D[i] < D[j]$ holds if $r(D[i]) < r(D[j])$, a notation that will get useful to translate the fact that $D[j]$ will be detected before $D[i]$ by algorithm DICT. As a consequence, the word of a sequence S that gets matched first is uniquely identified: it is $i = \arg \max_x (D[x] \mapsto S) = \max\{r(v) | v \in \max_{v' \in V_D} (v' \mapsto S)\}$: i denotes the maximal word in S .

Algorithm DICT distinguishes whether the subsequences next to a word $v \in V_D$ are empty (\emptyset) or chains (C), and we will refer to the subsequence before v by BF and to the subsequence after v by AF. Concretely, while recursively exploring a sequence between two already discovered parts $T_<$ and $T_>$ we check whether the maximal word v is directly next to $T_<$ (i.e., $T_< v, \dots, T_>$) or $T_>$ or both (\emptyset), or whether v is somewhere in the middle. In the latter case, we add a chain (C) to be able to find the greatest possible word in a next step.

DICT uses tuples of the form $(i, j, \text{BF}, \text{AF})$ where $i, j \in \mathbb{N}^2$ and $(\text{BF}, \text{AF}) \in \{\emptyset, C\}^2$, i.e., $D[i]$ denotes the maximal word in D , j is the number of consecutive occurrences of the corresponding word, and BF and AF represent the words before and after $D[i]$. These tuples are lexicographically ordered by the total order relation $>$ on the set of possible $(i, j, \text{BF}, \text{AF})$ tuples defined as follows: let $t = (i, j, \text{BF}, \text{AF})$ and $t' = (i', j', \text{BF}', \text{AF}')$ two such tuples. Then $t > t'$ iff $w > w'$ or $w = w' \wedge j > j'$ or $w = w' \wedge j = j' \wedge \text{BF} = C \wedge \text{BF}' = \emptyset$ or $w = w' \wedge j = j' \wedge \text{BF} = \text{BF}' \wedge \text{AF} = C \wedge \text{AF}' = \emptyset$.

With these definition we can prove that algorithm DICT is correct.

Theorem 1. *Given a dictionary for \mathcal{H} , algorithm DICT correctly discovers any $H \in \mathcal{H}$.*

Proof. We first prove that the claim is true if H forms a motif sequence (without edge expansion). Subsequently, we study the case where the motif sequence is expanded by Rule 2, and finally tackle the general composition case.

Discovery of motif sequences: Due to Lemma 4 it holds that for w chosen when Line 1 of *find_motif_sequence()* is executed for the first time, S is partitioned into three subsequences S_1 , w and S_2 . Subsequently *find_motif_sequence()* is executed on each of the subsequences $S' \in \{S_1, S_2\}$ recursively if $C \mapsto S'$, i.e., if the subsequences are not empty. Thus *find_motif_sequence()* computes a decomposition as described in Corollary 2 recursively. As each of the words used in the decomposition is

a subsequence of S and $\text{find_motif_sequence}()$ does not stop until no more words can be added to any subsequence, it holds that all nodes of S will be discovered eventually. In other words, $\pi^{-1}(u)$ is defined for all $u \in S$.

As a next step we assume $S' \neq S$ to be the sequence of words obtained by DICT to derive a contradiction. Since $S' := H'$ is the output of algorithm DICT and is hence embeddable in H : $S' \mapsto S$, there exists a valid embedding mapping π . Given $u, v \in V(S)$, we denote by $E^{\pi^{-1}}(S')$ the set of pairs $\{u, v\}$ for which $\{\pi^{-1}(u), \pi^{-1}(v)\} \in E(S')$. Now assume that S and S' do not lead to the same resource reservations “ $\pi(S) \neq \pi(S')$ ”. Hence there are some inconsistencies between the substrate and the output of algorithm DICT: $\Phi = \{\{u, v\} \in E(S) \setminus E^{\pi^{-1}}(S') \cup E^{\pi^{-1}}(S') \setminus E(S)\}$. With each of these “conflict” edges, one can associate the corresponding word $W_{u,v}$ (resp. $W'_{u,v}$) in S (resp. S'). If a given conflict edge spans multiple words, we only consider the words with the highest index as defined by DICT. We also define $i_{u,v} = r(W_{u,v})$ (resp. $i'_{u,v} = r(W'_{u,v})$). Since S' and S are by definition not isomorphic, $i'_{u,v} \neq i_{u,v}$.

Let $j = \max_{(u,v) \in \Phi} (i_{u,v})$ be the index of the greatest word embeddable on the substrate containing an inconsistency, and j' be the index of the corresponding word detected by DICT.

(i) Assume $j > j'$: a lower order motif was erroneously detected. Let J^+ (and J^-) be the set of dictionary entries that are detected before (after) $D[j]$ (if any) in S by DICT. Observe that the words in J^+ were perfectly detected by DICT, otherwise we are in Case (ii). We can decompose S as an alternating sequence of words of J^+ and other words using Corollary 2: $S = T_1 J_1(a_1) T_2 \dots T_k$ with $J_i(a_i) \in (J^+)^*$ and attachment points a_i and $T_i \in (J^-)^*$. As the words in J^+ are the same in S' , we can write $S' = T'_1 J_1 T'_2 \dots T'_k$ (using Corollary 2 as well).

Let T be the sequence among T_1, \dots, T_k that contains our misdeteected word $D[j]$, and T' the corresponding sequence in S' . Observe that $T' \mapsto T$ since the words J_i cut the sequences of S and S' into subsequences T_i, T'_i that are embeddable. Observe that $D[j] \mapsto T$ since T contains it. Note that in the execution of $\text{find_motif_sequence}()$ when $D[j']$ was detected the higher indexed words had been detected correctly by DICT in previous executions of this subroutine. Hence, $T_<$ and $T_>$ cannot contain any words leading to edges in Φ . Thus $(j', \dots, \dots) < (j, \dots, \dots)$ which contradicts Line 1 of $\text{find_motif_sequence}()$.

(ii) Now assume $j' > j$: a higher order motif was erroneously detected. Using the same decomposition as step (i), we define J'^+ as the set of words perfectly detected, and therefore decompose S and S' as sequences $S = T_1 J'_1 T_2 \dots J'_{k-1} T_k$ and $S' = T'_1 J'_1 T'_2 \dots J'_{k-1} T'_k$ with $J'_i \in (J'^+)^*$ and the property that each $T'_i \mapsto T_i$.

Let T' be the sequence among T'_1, \dots, T'_k that contains our misdeteected word $D[j']$, and T the corresponding sequence in S . Since $D[j'] \prec T'$, $D[j'] \mapsto T'$. Thus, since $T' \mapsto T$, we deduce $D[j'] \mapsto T$ which is a contradiction with j' and Corollary 2.

The same arguments can be applied recursively to show that conflicts in ϕ of smaller indices cannot exist either.

Expanded motif sequences. As a next step, we consider graphs that have been extended by applying node insertions (Rule 2) to motif sequences, so called *expanded* motif sequences: we prove that if H is an expanded motif sequence S , then algorithm DICT correctly discovers S . Given an expanded motif sequence S , replacing all two degree

nodes with an edge connecting their neighbors unless a cycle of length three would be destroyed, leads to a unique pure motif sequence T , $T \mapsto S$. For the corresponding embedding mapping π it holds that $V(S) \setminus \pi(T)$ is exactly the set \mathcal{R} of removed nodes. Applying *find_motif_sequence()* to an expanded motif sequence discovers this pure motif sequence T by using the nodes in \mathcal{R} as relay nodes. All nodes in \mathcal{R} are then discovered in *edge_expansion()* where the reverse operation node insertion is carried out as often as possible. It follows that each node in S is either discovered in *find_motif_sequence()* if it occurs in a motif or in *edge_expansion()* otherwise.

Combining expanded sequences. Finally, it remains to combine the expanded sequences. Clearly, since motifs describe all parts of the graph which are at least 2-connected, the graph remaining after collapsing motifs cannot contain any cycles: it is a tree. However, on this graph DICT behaves like TREE, but instead of attaching chains, entire sequences are attached to different nodes. Along the unique sequence paths between two nodes, DICT fixes the largest words first, and the claim follows by the same arguments as used in the proofs for tree and cactus graphs. \square

Algorithm 1 Motif Graph Discovery DICT

```

1:  $H' := \{\{v\}, \emptyset\}$  /*current request graph*/,  $\mathcal{P} := \{v\}$  /*set of unexplored nodes*/
2: while  $\mathcal{P} \neq \emptyset$  do
3:   choose  $v \in \mathcal{P}$ ,  $T := \text{find\_motif\_sequence}(v, \emptyset, \emptyset)$ 
4:   if  $(T \neq \emptyset)$  then  $H' := H' \vee T$ , add all nodes of  $T$  to  $\mathcal{P}$ , for all  $e \in T$  do edgeExpansion( $e$ )

5:   else remove  $v$  from  $\mathcal{P}$ 

find_motif_sequence( $v, T_<, T_>$ )
1: find maximal  $i, j, \text{BF}, \text{AF}$  s.t.  $H' \vee (T_<) \text{BF} (D[i])^j \text{AF} (T_>) \mapsto H$  where  $\text{BF}, \text{AF} \in \{\emptyset, C\}^2$ 
   /* issue requests */
2: if  $((i, j, \text{BF}, \text{AF}) = (0, 0, C, \emptyset))$  then return  $T_<CT_>$ 
3: if  $(\text{BF} = C)$  then  $\text{BF} = \text{find\_motif\_sequence}(v, T_<, (D[i])^j \text{AF} T_>)$ 
4: if  $(\text{AF} = C)$  then  $\text{AF} = \text{find\_motif\_sequence}(v, T_< \text{BF} (D[i])^j, T_>)$ 
5: return  $\text{BF} (D[i])^j \text{AF}$ 

edge_expansion( $e$ )
1: let  $u, v$  be the endpoints of edge  $e$ , remove  $e$  from  $H'$ 
2: find maximal  $j$  s.t.  $H' \vee C^j u \mapsto H$  /* issue requests */
3:  $H' := H' \vee C^j u$ , add newly discovered nodes to  $\mathcal{P}$ 

```

3.4 Request Complexity

The focus of DICT is on generality rather than performance, and indeed, the resulting request complexities can often be high. However, as we will see, there are interesting graph classes which can be solved efficiently.

Let us start with a general complexity analysis. The requests issued by DICT are constructed in Line 1 of *finding_motif_sequence()* and in Line 2 of *edge_expansion()*. We will show that the request complexity of the latter is linear in the number of edges of the host graph while the request complexity of *finding_motif_sequence()* depends on the structure of the dictionary. Essentially, an

efficient implementation of Line 1 of *finding_motif_sequence* in DICT can be seen as the depth-first exploration of the dictionary D starting from the chain C . More precisely, at a dictionary word v requests are issued to see if one of the outgoing neighbors of v could be embedded at the position of v . As soon as one of the replies is positive, we follow the corresponding edge and continue recursively from there, until no outgoing neighbors can be embedded. Thus, the number of requests issued before we reach a vertex v can be determined easily.

Recall that DICT tests vertices of a dictionary D according to a fixed port labeling scheme. For any $v \in V_D$, let $p(C, v)$ be the set of paths from C to v (each path including C and v). In the worst case, discovering v costs $cost(v) = \max_{p \in p(C, v)} (\sum_{u \in p} |\Gamma^+(u)|)$.

Lemma 6. *The request complexity of Line 1 of $find_motif_sequence(v', T_<, T_>)$ to find the maximal $i, j, \text{BF}, \text{AF}$ such that $H'v' (T_<) \text{BF} (D[i])^j \text{AF} (T_>) \mapsto H$ where $\text{BF}, \text{AF} \in \{\emptyset, C\}^2$ and H' is the current request graph is $O(\max_{v \in V_D} cost(v) + j)$.*

Proof. To reach a word $v = D[i]$ in V_D with depth-first traversal there is exactly one path between the chain C and v . DICT issues a request for at most all the outgoing neighbors of the nodes this path. After v has been found, the highest j where $H'v (T_<) \text{BF} (v^j) \text{AF} (T_>) \mapsto H$ has to be determined. To this end, another $j + 1$ requests are necessary. Thus the maximum of $cost(v) + j$ over all word $v \in V_D$ determines the request complexity. \square

When additional nodes are discovered by a positive reply to an embedding request, then the request complexity between this and the last previous positive reply can be amortized among the newly discovered nodes. Let $num_nodes(v)$ denote the number of nodes in the motif sequence of the node v in the dictionary.

Theorem 2. *The request complexity of algorithm DICT is at most $O(n \cdot \Delta + m)$, where m denotes the number of edges of the inferred graph $H \in \mathcal{H}$, and Δ is the maximal ratio between the cost of discovering a word v in D and $num_nodes(v)$, i.e., $\Delta = \max_{v \in V_D} \{cost(v)/num_nodes(v)\}$.*

Proof. Each time Line 1 of *find_motif_sequence()* is called, either at least one new node is found or no other node can be embedded between the current sequences (one request is necessary for the latter result). If one or more new nodes are discovered, the request complexity can be amortized by the number of nodes found: If v is the maximal word found in Line 1 of *find_motif_sequence()* then it is responsible for at most $cost(v)$ requests due to Lemma 6. If it occurs more than once at this position, only one additional request is necessary to discover even more nodes (plus one superfluous request if no more occurrences of v can be embedded there). Amortizing the request number over the number of discovered nodes results in Δ requests. All other requests are due to *edge_expansion(e)* where additional nodes are placed along edges. Clearly, these costs can be amortized by the number of edges in H : for each edge $e \in E(H)$, at most two embedding requests are performed (including a “superfluous” request which is needed for termination when no additional nodes can be added). \square

3.5 Examples

Let us consider concrete examples to provide some intuition for Theorem 1 and Theorem 2. The execution of DICT for the graph in Figure 2.b), is illustrated in Figure 3.

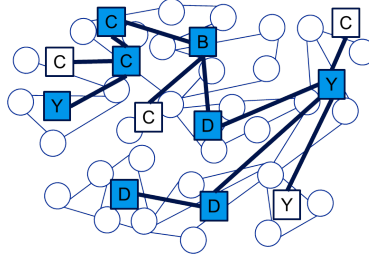


Fig. 3. Motif sequence tree of the graph in Figure 2 b). The squares and the edges between them depict the motif composition, the shaded squares belong to the motif sequence YC^2BDYD^2 discovered in the first execution of *find_motif_sequence()* (chains, cycles, diamonds, and the complete bipartite graph over two times three nodes are denoted by C , Y , D and B respectively). Subsequently, the found edges are expanded before calling *find_motif_sequence()* another four times to find Y and three times C .

A fundamental graph class are *trees*. Since, the tree does not contain any 2-connected structures, it can be described by a single motif: the chain C . Indeed, if DICT is executed with a dictionary consisting in the singleton motif set $\{C\}$, it is equivalent to a recursive version of TREE from [11] and seeks to compute maximal paths. For the cactus graph, we have two motifs, the request complexity is the same as for the algorithm described in [11].

Corollary 3. *Trees can be described by one motif (the chain C), and cactus graphs by two motifs (the chain C and the cycle Y). The request complexity of DICT on trees and cactus graphs is $O(n)$.*

Proof. We present the arguments for cactus graphs only, as trees constitute a subset of the cactus family. The absence of diamond graph minors implies that a cactus graph does not contain two closed faces which share a link. Thus, there can exist at most two different (not even disjoint) paths between any node pair, and the corresponding motif subgraph forms a *cycle* Y (or a triangle). Since the cycle has only one attachment point pair, Δ of D is constant. Consequently, a linear request complexity follows directly from Theorem 2 due to the planarity of cactus graphs (i.e., $m \in O(n)$). \square

An example where the dictionary is efficient although the connectivity of the topology can be high are *block graphs*. A block graph is an undirected graph in which every bi-connected component (a *block*) is a clique. A *generalized block graph* is a block graph where the edges of the cliques can contain additional nodes. In other words, in the terminology of our framework, the motifs of generalized block graphs are *cliques*. For instance, cactus graphs are generalized block graphs where the maximal clique size is three.

Corollary 4. Generalized block graphs can be described by the motif set of cliques. The request complexity of DICT on generalized block graphs is $O(m)$, where m denotes the number of edges in the host graph.

Proof. The framework dictionary for generalized block graphs consists of the set of cliques, as a clique with k nodes cannot be embedded on sequences of cliques with less than k nodes. As there are three attachment point pairs for each complete graph with four or more nodes, DICT can be applied using a dictionary that contains three entries for each motif with more than three nodes ($num_nodes() > 3$). Thus, the i^{th} dictionary entry has $\lfloor i/3 \rfloor + 3$ nodes for $i > 1$ and $cost(D[i]) < 3(i + 2)$ and Δ of D is hence in $O(1)$. Consequently the complexity for generalized block graphs is $O(m)$ due to Theorem 2. \square

On the other hand, Theorem 2 also states that highly connected graphs may require $\Omega(n^2)$ requests, even if the dictionary is small. In the next section, we will study whether this happens in “real world graphs”.

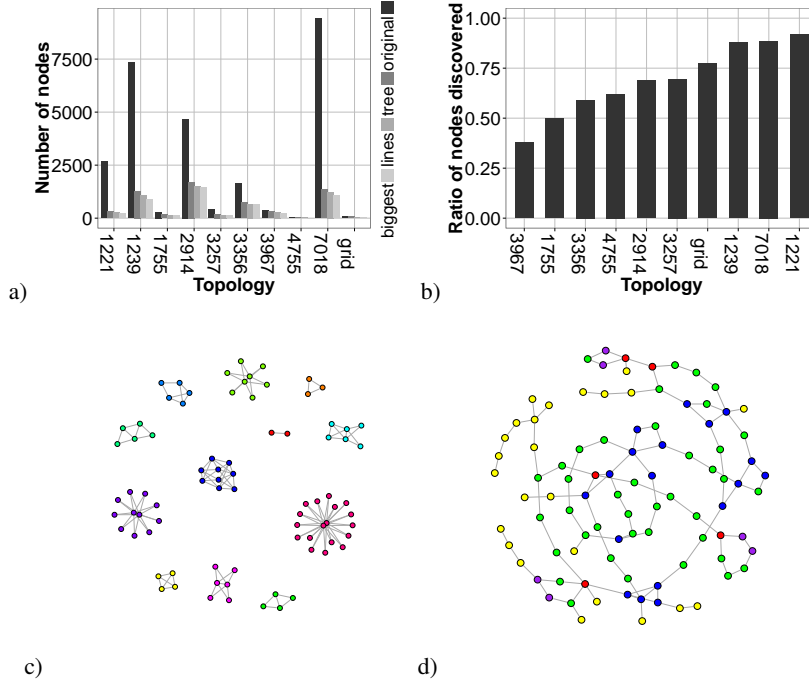


Fig. 4. Results of DICT when run on different Internet and power grid topologies. a) Number of nodes in different autonomous systems (AS). We computed the set of motifs of these graphs as described in Definition 2 and counted the number of nodes that: (i) belong to a tree structure at the fringe of the network, (ii) have degree 2 and belong to two-connected motifs, and finally (iii) are part of the largest motif. b) The fraction of nodes that can be discovered with 12-motif dictionary represented in Figure c). d) An example network where tree nodes are colored yellow, line-nodes are green, attachment point nodes are red and the remaining nodes blue.

4 Experiments

To complement our theoretical results and to validate our framework on realistic graphs, we dissected the *ISP topologies* provided by the Rocketfuel mapping engine¹. In addition, we also dissected the topology of a European electricity distribution grid (`grid` on the legends). Figure 4 a) provides some statistics about the aforementioned topologies. Since DICT discovers both tree and degree 2 nodes in linear time, this figure shows that most of each topology can be discovered quickly. The inspected topologies are composed of a large bi-connected component (the largest motif), and some other small and simple motifs. Figure 4 b) represents the fraction of each topology that can be discovered by DICT using only a 12-motifs dictionary (see Figure 4 c)). Interestingly, this small dictionary is efficient on 10 different topologies, and contains motifs that are mostly symmetrical. This might stem from the man-engineered origin of the targeted topologies. Finally, Figure 4 d) provides an example of such a topology.

References

1. H. Acharya and M. Gouda. On the hardness of topology inference. In *Proc. ICDCN*, pages 251–262, 2011.
2. A. Anandkumar, A. Hassidim, and J. Kelner. Topology discovery of sparse random graphs with few participants. In *Proc. SIGMETRICS*, 2011.
3. N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer. Minimum congestion mapping in a cloud. In *Proc. 30th PODC*, pages 267–276, 2011.
4. B. Cheswick, H. Burch, and S. Branigan. Mapping and visualizing the internet. In *Proc. USENIX Annual Technical Conference (ATEC)*, 2000.
5. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Elsevier Computer Networks*, 54(5), 2010.
6. G. Even, M. Medina, G. Schaffrath, and S. Schmid. Competitive and deterministic embeddings of virtual networks. In *Proc. ICDCN*, 2012.
7. J. Fan and M. H. Ammar. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *Proc. IEEE INFOCOM*, 2006.
8. I. Houidi, W. Louati, and D. Zeghlache. A distributed virtual network mapping algorithm. In *Proc. IEEE ICC*, 2008.
9. J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. ACM SIGCOMM VISA*, 2009.
10. Y. A. Pignolet, G. Tredan, and S. Schmid. Misleading Stars: What Cannot Be Measured in the Internet? In *Proc. DISC*, 2011.
11. Y.-A. Pignolet, G. Tredan, and S. Schmid. Adversarial VNet Embeddings: A Threat for ISPs? In *IEEE INFOCOM*, 2013.
12. G. Schaffrath, S. Schmid, and A. Feldmann. Optimizing long-lived cloudnets with migrations. In *Proc. IEEE/ACM UCC*, 2012.
13. B. Yao, R. Viswanathan, F. Chang, and D. Waddington. Topology inference in the presence of anonymous routers. In *Proc. IEEE INFOCOM*, pages 353–363, 2003.
14. S. Zhang, Z. Qian, J. Wu, and S. Lu. An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In *Proc. IEEE INFOCOM*, 2012.

¹ See <http://www.cs.washington.edu/research/networking/rocketfuel/>.

A Appendix

Lemma 1. *The embedding relation \mapsto applied to any family \mathcal{G} of undirected graphs (short: (\mathcal{G}, \mapsto)), forms a partially ordered set (a poset).*

Proof. A poset structure (S, \preceq) over a set S requires that \preceq is a (*reflexive*, *transitive*, and *antisymmetric*) order which may or may not be partial. To show that (\mathcal{G}, \mapsto) , the embedding order defined over a given set of graphs \mathcal{G} , is a poset, we examine the three properties in turn.

Reflexive $G \in \mathcal{G} \mapsto G \in \mathcal{G}$: By using the identity mapping $\pi : G = (V, E) \rightarrow G = (V, E)$ which embeds each node and link to itself, the claim is proved.

Transitive $A \in \mathcal{G} \mapsto B \in \mathcal{G}$ and $B \in \mathcal{G} \mapsto C \in \mathcal{G}$ implies $A \in \mathcal{G} \mapsto C \in \mathcal{G}$: Let π_1 denote the embedding function for $A \in \mathcal{G} \mapsto B \in \mathcal{G}$ and let π_2 denote the embedding function for $B \in \mathcal{G} \mapsto C \in \mathcal{G}$, which must exist by our assumptions. We will show that then also a valid embedding function π exists to map A to C . Regarding the node mapping, we define π_V as $\pi_V := \pi_{1V} \circ \pi_{2V}$, i.e., the result of first mapping the nodes according to π_{1V} and subsequently according to π_{2V} . We first show that π_V is a valid mapping from A to C as well. First, $\forall v_A \in V_A$, $\pi(v_A)$ maps v_A to a single node in V_C , fulfilling the first condition of the embedding (see Definition 1). Ignoring relay capacities (which is studied together with the conditions on the links below), Condition (ii) of Definition 1 is also fulfilled since the mapping π_{1V} ensures that no node in V_B exceeds its capacity, and can hence safely be mapped to V_C . Let us now turn our attention to the links. We use the following mapping π_E for the edges. Note that π_{1E} maps a single link e to an entire (but possibly empty) path in B and π_{2E} then maps the corresponding links e' in B to a walk in C . We can transform any of these walks into paths by removing cycles; this can only lower the resource costs. Since π_{1E} maps to a subset of E_B only and since π_{2E} can embed all edges of B , all link capacities are respected up to relay costs. However, note also that by the mapping π_1 and for relay costs $\epsilon > 0$, each node $v_B \in V_B$ can either not be used at all, be fully used as a single endpoint of a link $e_A \in E_A$, or serve as a relay for one or more links. Since both end-nodes and relay nodes are mapped to separate nodes in C , capacities are respected as well. Conditions (iii) and (iv) hold as well.

Antisymmetric $A \in \mathcal{G} \mapsto B \in \mathcal{G}$ and $B \in \mathcal{G} \mapsto A \in \mathcal{G}$ implies $A = B$, i.e., A and B are isomorphic and have the same weights: First observe that if the two networks differ in size, i.e., $|V_A| \neq |V_B|$ or $|E_A| \neq |E_B|$, then they cannot be embedded to each other: W.l.o.g., assume $|V_A| > |V_B|$, then since nodes of V_A cannot be split into multiple nodes of V_B (cf Definition 1), there exists a node $v_A \in V_A$ to which no node from V_B is mapped. This however implies that node $\pi_1(v_A) \in V_B$ must have available capacities to host also v_A , contradicting our assumption that nodes cannot be split in the embedding. Similarly, if $|E_A| \neq |E_B|$, we can obtain a contradiction with the single path argument. Thus, not only the total number of nodes and links in A and B must be equivalent but also the total amount of node and link resources. So consider a valid embedding π_1 for $A \in \mathcal{G} \mapsto B \in \mathcal{G}$ and a valid embedding π_2 for $B \in \mathcal{G} \mapsto A \in \mathcal{G}$, and assume $|V_A| = |V_B|$ and $|E_A| = |E_B|$. It holds that π_1 and π_2 define an isomorphism between A and B : Clearly, since $|V_A| = |V_B|$, π_1 and π_2 define a permutation on the vertices. W.l.o.g., consider any link $\{v_A, v'_A\} \in E_A$. Then,

also $\{\pi_1(v_A), \pi_1(v'_A)\} \in E_B$: $|\{\pi_1(v_A), \pi_1(v'_A)\}| = 0$ would violate the node capacity constraints in B , and $|\{\pi_1(v_A), \pi_1(v'_A)\}| > 1$ requires $|E_B| > |E_A|$. \square

Lemma 5. *There exists a dictionary $D = (V_D, E_D)$ that covers all member graphs H of a motif graph family \mathcal{H} with n vertices.*

Proof. We present a procedure to construct such a dictionary D . Let \mathcal{M}_n be the set of all motifs with n nodes of the graph family \mathcal{H} . For each motif $m \in \mathcal{M}_n$ with x possible attachment point pairs (up to isomorphisms), we add x dictionary words to V_D , one for each attachment point pair. The resulting set is denoted by V_M . For each sequence of V_M^* with at most n nodes, we add another word to V_D (with the un-used attachment points of the first and the last subword). There is an edge $e \in E_D$ if the transitive reduction of the embedding relation with context includes an edge between two words. We now prove that D is a dictionary, i.e., it is robust to composition. Let $i \in V_D$. Observe that R_i contains all compositions of words with at most n nodes in which i can be embedded. Consequently, no matter which sequences are in \overline{R}_i^* it holds that v_i cannot be embedded in a sequences in Q_i the robustness condition is satisfied. Since H has n vertices, and since D contains all possible motifs of at most n vertices, D covers H . \square

Note that the proof of Lemma 5 only addresses the composition robustness for sequences of up to n nodes. However, it is clear that $|V(G)| > |V(H)| \Rightarrow G \not\rightarrow H$, and therefore no “mismatch” can happen to happen. Finite dictionaries and with this adapted composition can also be applied in the lemmata proved above, there is only a small notational change necessary in the proof of Lemma 4. (Note that it is always possible to determine the number of nodes n by binary search using $O(\log n)$ requests.)